2. TECHNICAL ARCHITECTURE OVERVIEW

This section provides an overview of technical architecture, as well as the services, components, and distribution strategies that comprise a technical architecture. Subsection 2.1 defines technical architecture types and Subsection 2.2 describes technical architecture services and strategies.

2.1 Architecture Types

Technical architectures represent detailed plans from which systems can be implemented. Just as requirements articulate the end users' vision of system functionality and performance, technical architectures define the technical and organizational vision for a system. Technical architectures can be classified into three broad types that correspond to three different levels of technical detail and specificity.

- **Framework Architecture**: Based on high-level functional requirements, framework architectures define the overall process and data distribution strategy to be used by the system.

- **Conceptual Architecture**: Based upon a framework architecture, a conceptual architecture describes how system requirements are allocated to physical components. The conceptual architecture is tightly coupled to specific, detailed functional requirements. Additionally, the conceptual architecture refines the component distribution model proposed within the framework architecture and defines specific component resource needs based on allocated requirements. The conceptual architecture also recommends specific standards with which the architecture should comply, and makes an initial recommendation of products to implement architecture components.

- **Target Architecture**: Based upon a conceptual architecture and upon preliminary system design activities, the target architecture provides a formal and detailed plan from which a system can be implemented. The target architecture refines identified system components, component distribution strategies, and estimated component resource needs. The target architecture is tightly coupled with a system's preliminary designs and is used as a blueprint for migrating, implementing, and maintaining a system.

Architectures evolve in parallel with system designs, encompassing the following activities:

Step 1: Identify preliminary system requirements, and organizational and technical goals.
Step 2: Define a framework technical architecture that is most appropriate for satisfying target system requirements and achieving organizational and technical goals.
Step 3: Refine system distribution strategies by allocating functional and data requirements to framework architecture.
Step 4: Define a conceptual architecture that is most appropriate for the system distribution strategy.
Step 5: Determine system design strategy based on conceptual architecture, performance (e.g., reliability, accessibility, response time) requirements, and refined data and transaction volume estimates.
Step 6: Define a target architecture most appropriate for the system design strategy.

Figure 2-1 illustrates these steps, and that the target architecture is a tailored augmentation of the conceptual architecture, which is based on the framework architecture. Additionally, Figure 2-1 illustrates that architecture designs are dependent upon:

- System requirements, which result from business area analysis activities.
- System distribution strategies, which result from process and data distribution analysis.
- System design strategy, which results from preliminary design activities.

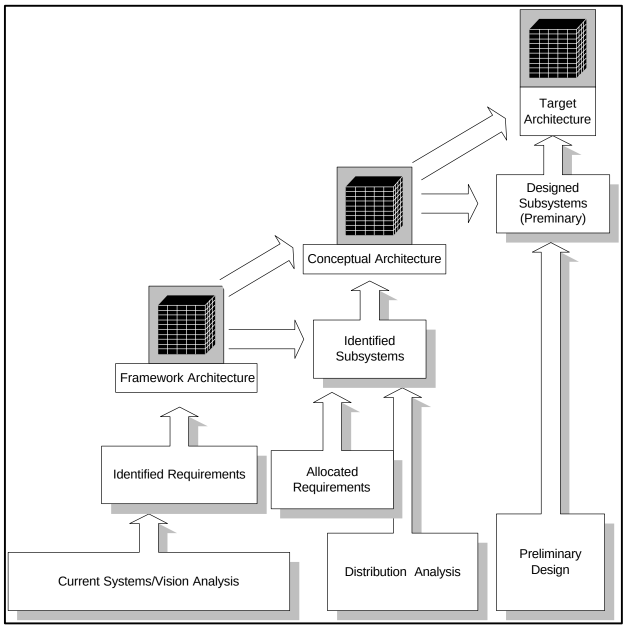

Figure 2-1. Evolution of an Architecture

Figure 2-2 summarizes the activities involved in defining each of the architectural types:

	Architecture Services Identified	Significant System Components Identified	Component Distribution Proposed	Requirements Allocated to Components	Component Distribution Refined	Components Sized Based on Allocated Requirements	Requirements Allocation to Components Refined	Component Distribution Finalized	Component Sizing Refined
Target Architecture							√	√	√
Conceptual Architecture				√	√	√			
Framework Architecture	√	√	√						

Figure 2-2. Building Blocks of an Architecture

Technical architectures describe the components (e.g. hardware, software, telecommunications) that comprise an information system and describe the interrelationships among these components. Technical architectures also:

- Serve as a framework for understanding and standardizing systems and their components.
- Provide models for understanding both existing and future systems.
- Define frameworks for satisfying integrated system requirements.
- Decrease interface-related complexity, defects, and time to market.
- Increase product and process predictability during system development.
- Facilitate the integration of COTS and other standardized components into a system.

To provide the flexibility required by changing political and legal climate, economic pressures, time-to-market requirements, technological improvements, and other factors, system architectures should:

- Provide easy, transparent, and secure access to organizational data.
- Support rapid deployment of new application systems.
- Support rapid modification of existing application systems.
- Support integration of new and existing application systems.
- Support current and emerging technologies and standards.
- Support dynamic reconfiguration of systems for scalability or network requirements.
- Support interoperability, manageability, reliability, availability, and security requirements.

As Figure 2-3 illustrates, architectures "frame" a system in terms of components (hardware, software, and networking infrastructure) and interconnections (interfaces) among these components. Working together, architectural components provide the services required to satisfy an organization's needs.

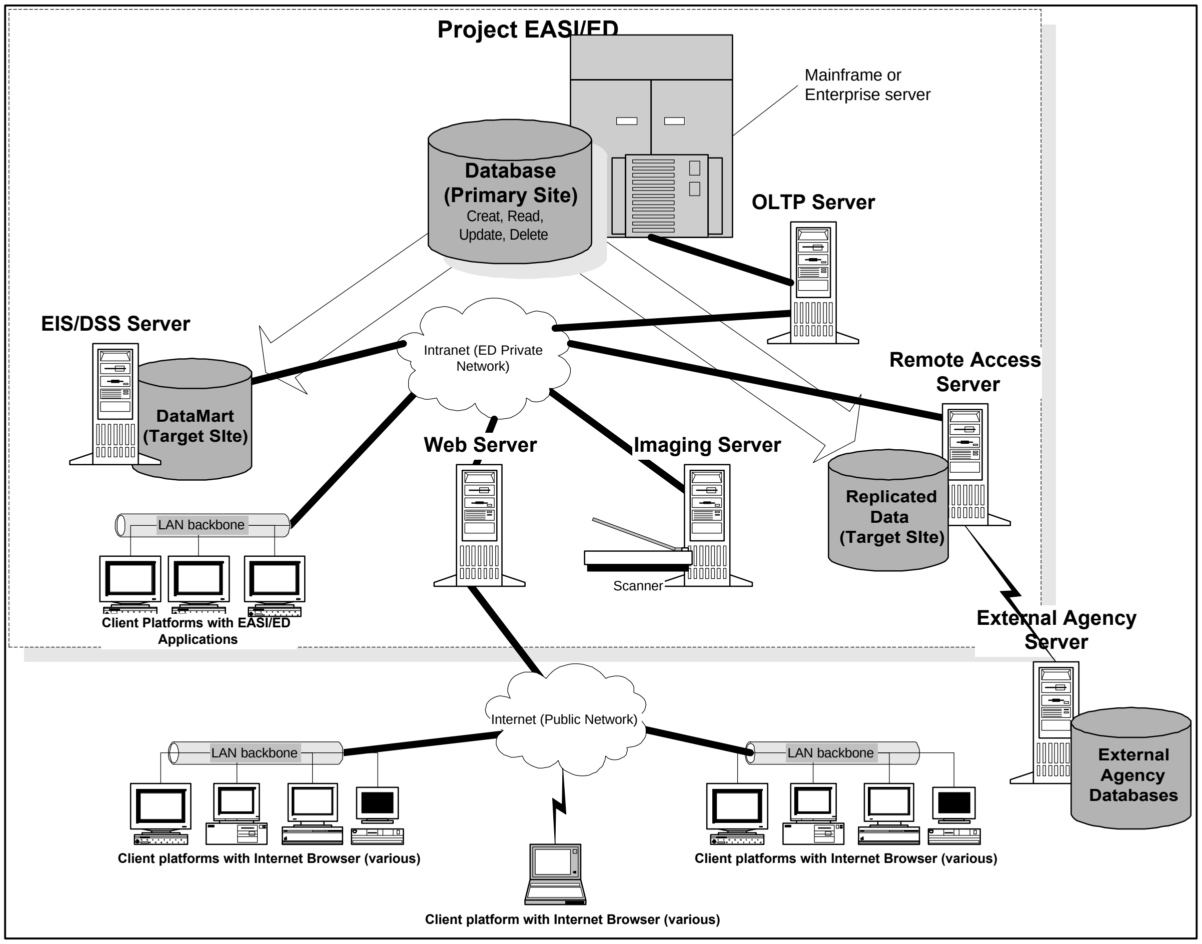
Project EASI/ED
Mainframe or Enterprise server
Database (Primary Site)
Creat, Read, Update, Delete
OLTP Server
EIS/DSS Server
DataMart (Target SIte)
Intranet (ED Private Network)
Remote Access Server
Web Server
Imaging Server
Replicated Data (Target SIte)
LAN backbone
Client Platforms with EASI/ED Applications
Scanner
External Agency Server
Internet (Public Network)
LAN backbone
LAN backbone
External Agency Databases
Client platforms with Internet Browser (various)
Client platforms with Internet Browser (various)
Client platform with Internet Browser (various)

Figure 2-3. Components of a sample Architecture

2.2 Technical Architecture Services

Technical architecture services define the capabilities delivered by an architecture and facilitate organization of the architecture. Services are implemented within architectures via architectural components. As Figure 2-3 illustrates, technical architecture components may include hardware platforms (such as database or specialized application servers), telecommunication technologies (such as those used to implement local and wide area networks), and other infrastructure technologies. Additionally, technical architectures include system and application software. This software comprises COTS and custom developed applications. System architectures are defined in terms of selected components (hardware, software, etc.), component processing strategies, and the interconnection among components.

Architecture services are the capabilities required to satisfy system requirements. Using information about the existing system(s) and about new organizational and technical requirements, architects define the services to be delivered by the new system. Using the service definitions, system architects select the most appropriate architectural components and allocate requirements to those components that are most suitable for delivering specific required functionality and performance. This process is illustrated in Figure 2-4.

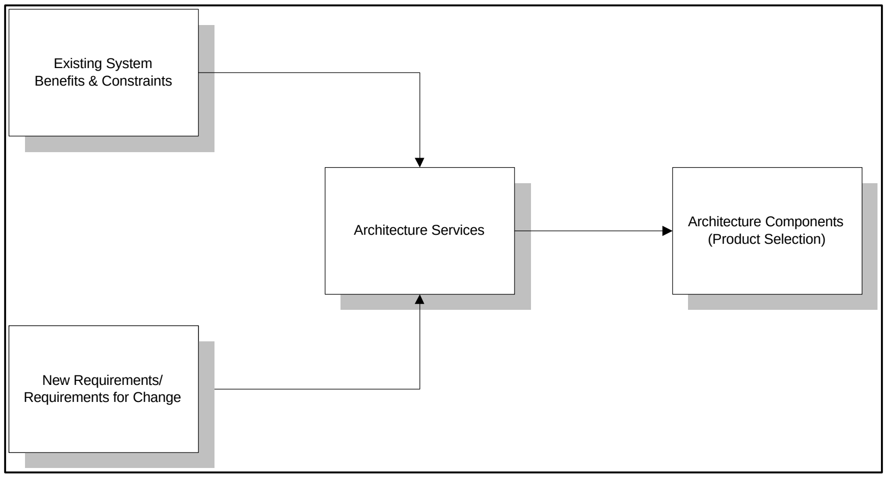

Figure 2-4. Process for Defining Architecture Components

As Figure 2-5 illustrates, architecture services can be classified as:

- **Presentation services** that provide the mechanisms through which users interact with a system. Presentation services include screen generation, window management, and on-line help.

- **Application services** that provide mechanisms through which business logic is implemented. Application services govern business functions and processes performed by an application. These services are typically invoked via the presentation services when a user issues a request or by other business services. Application services may be computationally significant and may include transaction processing, edit and validation, message queuing, data typing and conversion, and process management.

- **Data management services** that provide mechanisms through which data is accessed and managed (created, updated, read, and deleted). Data management services include database management, file management, and related services.

- **Infrastructure services** that provide common mechanisms for improved communication efficiency and coordination among system components. Infrastructure services include distributed time, file, name/directory, and security services.

- **System management services** that provide mechanisms for managing system resources, including data. System management services include archive management, software distribution, system and performance monitoring, problem reporting, and hardware/software asset management.

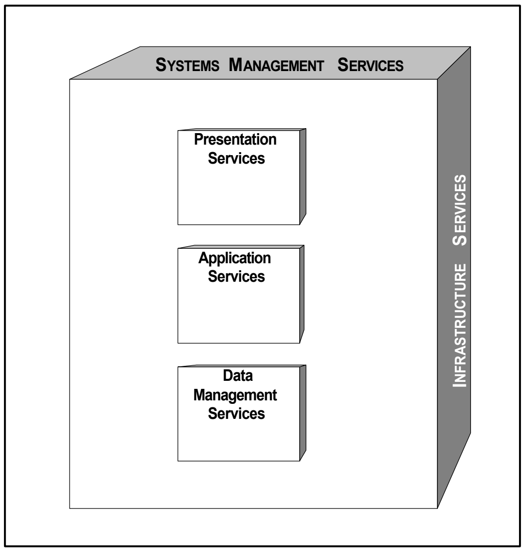

Figure 2-5. Classification of Architecture Services

2.3 Technical Architecture Processing and Distribution Strategies

Components within a technical architecture may follow a number of different processing and distribution strategies. Processing strategies define the way(s) in which users may access and change data in the system, while distribution strategies define the way in which software components and data resources are distributed within the system.

Processing strategies commonly used to describe system architectures are:

• **Batch**	The user has no immediate capability to change system data. This model allows batch file updating, as well as batch data entry, validation, and collection. Data is processed at scheduled intervals. On-line inquiry capabilities are not available.
• **Batch with Online Data Collection**	The user has no immediate capability to change system data. This model allows batch file updating and on-line data entry/collection. Transactions are entered and transferred to either a tape or disk file. Later, the transactions are edited and stored in a posting file for subsequent batch file update of system data.
• **On-line Inquiry**	The user has no immediate capability to change system data. This model allows on-line inquiry with batch data entry and file updating. The user can access the computer and review data managed by the system (as of the last update), but cannot immediately modify system data. To modify data managed by the system, batch processes must be used.

- **On-line Inquiry and Data Collection** The user has no immediate capability to change system data. This model allows on-line inquiry, batch file updating, and on-line data entry, data validation and data collection. Transactions are entered and transferred to either a tape or disk file. Or at the time of entry later, the transactions are edited and stored in a posting file for subsequent batch file update of system data.

- **Real-Time Control** The user has the capability to immediately change system data. This model allows on-line inquiry, data entry, and file updating. The user uses directly updates the computer files by entering one transaction at a time. In such a system, there may not be any batch controls.

- **Remote Job Entry** The user has no immediate capability to change system data. Data from remote locations is entered in batches, e.g., the operator enters a batch update, which is reconciled and processed at a scheduled time.

These processing strategies significantly affect the selection of architecture components. However, the interconnection of these components is largely determined by the distribution of processes (i.e. software components) and data. For this reason, three classes of distribution strategies are considered when developing system architectures. These classes are:

- **Physical software (process) distribution strategies** that define how software components, providing presentation, application, and data management services, will be allocated to system hardware.

- **Logical software (process) distribution strategies** that define how system functionality will be partitioned among software components. For example, functionality for providing application services in a system may be physically located on a single hardware component, but be logically segmented into separate software components, with a set of interfaces among these components.

- **Data Distribution strategies** that define how data will be distributed throughout a system.

As illustrated in Figure 2-6, these strategies are used to define technical architectures. Framework technical architectures, are primarily based on the physical software (process) and data distribution strategies. The conceptual and target technical architectures are based on the framework architecture and on the logical software (process) distribution strategy.

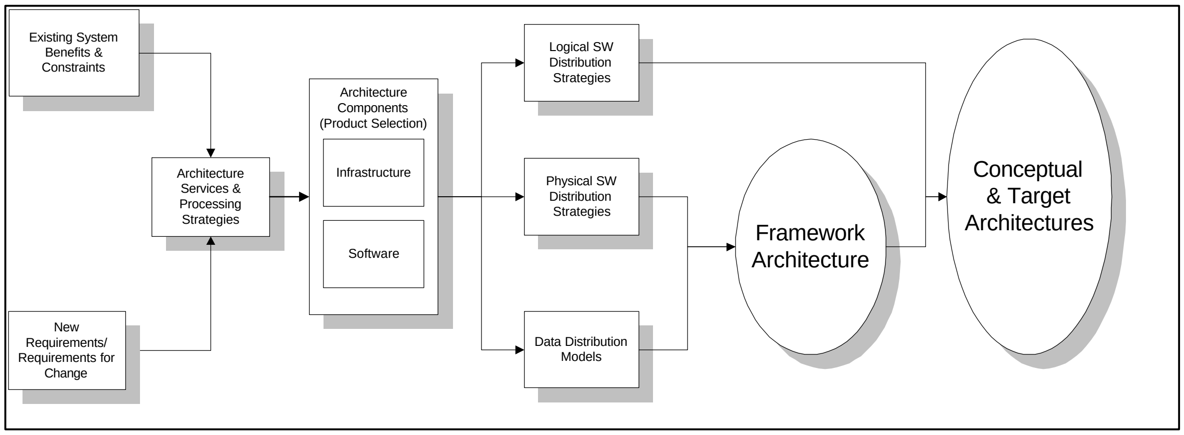
Existing System Benefits & Constraints
New Requirements/ Requirements for Change
Architecture Services & Processing Strategies
Architecture Components (Product Selection)
Infrastructure
Software
Logical SW Distribution Strategies
Physical SW Distribution Strategies
Data Distribution Models
Framework Architecture
Conceptual & Target Architectures

Figure 2-6. Process for Deriving Framework, Conceptual, and Target Architectures

2.3.1 Physical Software (Process) Distribution Strategies

Six physical software distribution strategies are commonly used to describe architectures:

• **Monolithic**	All logic – including presentation, application, and data management – is performed on the server side of the network. The client side of the network, where the user is located, is typically a dumb terminal (VT100 or 3270). This software distribution strategy is also referred to as “centralized processing.”
• **Distributed Presentation**	Almost the entire user interface is located on the server side of the network. The presentation management subsystem resides on the client. Part of the logic associated with end-user presentation resides on the client side of network, part on the server. The application logic and the database reside on the server side of the network. “Screen scraping” is an example of this model –e.g., the server prepares a terminal data stream intended for a dumb terminal, but the client does not display it in raw form. Rather, the client extracts data fields and creates a new interface for the user.
• **Remote Presentation**	Application code and data management software executes on the server. All software used for screen painting, including the application logic for presentation and presentation management, resides on the client side of network. This distribution model is sometimes referred to as “thin client.” Systems using PC X Server-based clients are examples of this model.
• **Distributed Logic**	Application logic is split across the server and client sides of the network. The application code on the server side is normally related to data input and output (I/O). This code may be system-specific stored procedures, transaction-processing logic, or application logic used throughout an enterprise. Application code associated with user I/O resides on the client side of the network.
• **Remote Data Management**	Application and data management functions are physically separated. The user interface and application are on the client side of network, the data management software and data reside on the server side of the network. This distribution model is sometimes referred to as “fat client.” This is currently the most commonly used physical distribution model.
• **Distributed Data Management**	System data management software and data reside on both the client and server sides of the network.

Physical software distribution strategies are used to allocate software components (i.e. processes) to hardware resources. This physical distribution describes elementary relationships between software components and the hardware to which these components are allocated. These strategies, (illustrated in Figure 2-7) describe the physical distribution or “tiers” of software within an architecture. However, the physical tiers identified within these strategies do not describe how software components will be logically organized, nor do they describe distributed component interfaces or communication mechanisms.

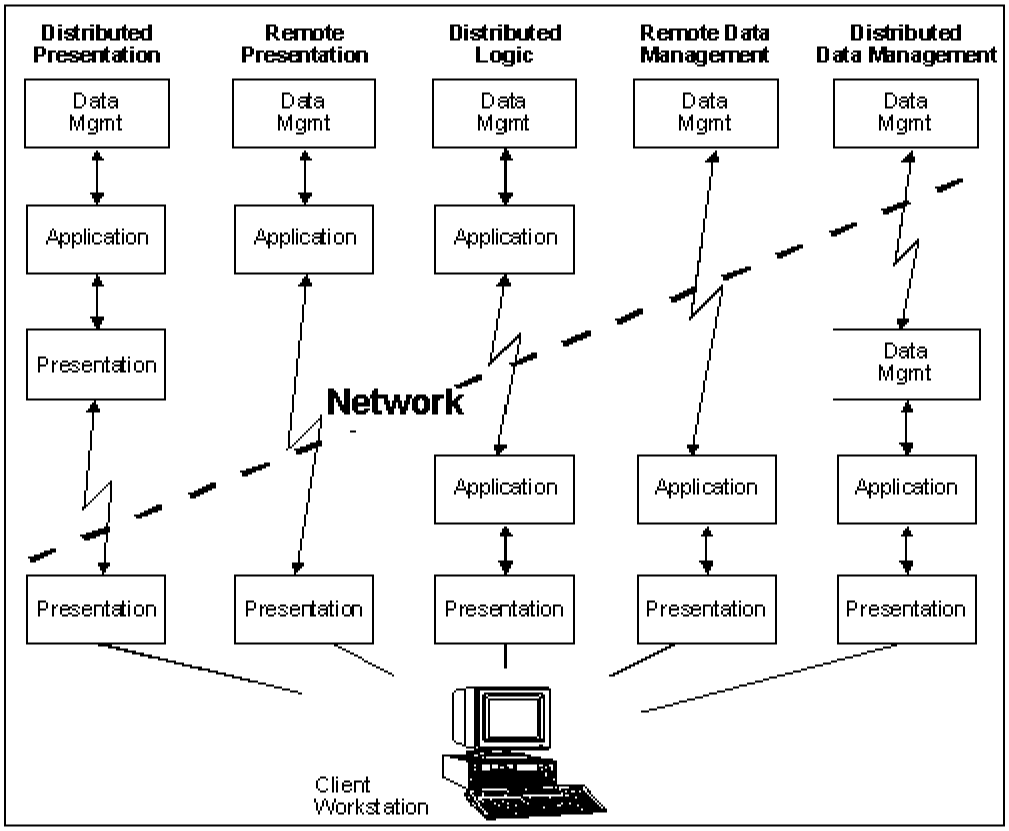

Figure 2-7. Physical Software Distribution

Many complex system architectures simultaneously employ several of the physical distribution strategies described in this subsection. Nonetheless, architectures can be generally classified as centralized (monolithic) or as distributed.

2.3.2 Logical Software (Process) Distribution Strategies

Figure 2-8 shows four ways in which software components may be distributed across multiple physical tiers. Within each physical tier, software components may also be distributed logically. This involves segmenting software components and defining the interfaces between them. For example, in Figure 2-8, in the case where business and data access logic is physically located on Tier 2 (a specific hardware component), it is possible to logically distribute this functionality into two separate software components (one for business logic and another for data access logic).

Unlike physical software distribution strategies, logical distribution strategies do not describe where software will be physically located within an architecture. Rather, logical distribution strategies articulate software design characteristics and define:

- Logical software "tiers," which describe how software components will be segmented or partitioned.
- Interfaces between partitioned software components.

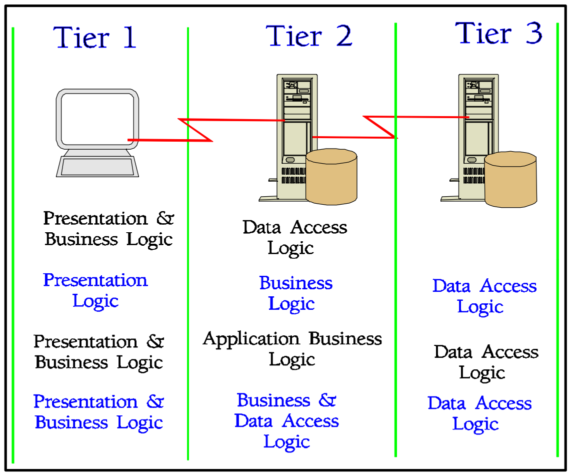
Tier 1
Tier 2
Tier 3
Presentation & Business Logic
Data Access Logic
Presentation Logic
Business Logic
Data Access Logic
Presentation & Business Logic
Application Business Logic
Data Access Logic
Presentation & Business Logic
Business & Data Access Logic
Data Access Logic

Figure 2-8. Logical Software Distribution

Software designs are significantly affected by the logical software distribution strategy employed within an architecture. As a result, when selecting logical distribution strategies, traditional software engineering concepts must be considered. Architecturally important software engineering considerations include:

- **Component Coupling** – refers to the independence of software components. In a narrow sense, coupling refers to the way data is exchanged between components. Loose coupling is generally better than tight coupling. The loosest, and therefore preferred, type of coupling is data coupling, where data is transferred as parameters via well-defined interfaces. The tightest, or least desirable, coupling involves components directly referencing shared variables. Tight coupling often indicates that components are not insulated from each other, and are not designed to be separate and independent. Tightly coupled components are usually complex, difficult to maintain, and monolithic. As a result there is very little flexibility regarding physical distribution of components. Two applications that communicate with each other via database management system (DBMS) updates, but which are otherwise independent of each other, would be considered loosely coupled.

- **Component Cohesion** – refers to component conceptual or semantic coherence. Cohesion reflects the degree to which one component implements one function or a group of similar functions. For example, cohesive components do not implement multiple, disparate services, such as presentation and application logic. Highly cohesive components are typically more understandable and thus easier to maintain. Additionally, cohesion promotes logical and physical software distribution flexibility, which in turn promotes system scalability. An application composed of logically separate presentation, application, and data management components, would be considered highly cohesive.

Software coupling and cohesion directly affect software modularity and interface design. As a result, coupling and cohesion directly affect the flexibility and complexity of software architectures. For example, when software component interfaces are based on widely accepted standards, as illustrated in Figure 2-9, logical software tiers can promote component interoperability and substitutability. That is, logical tiers can allow components within one tier to be changed without affecting other tiers. Loosely coupled and highly cohesive software architectures introduce flexibility regarding the physical distribution of software components. For example, logical distribution strategies may allow components residing on one computer to be relocated without introducing significant design complications. Figure 2-10 illustrates how the logical separation of software components can allow physical distribution of these components to be changed without dramatically affecting the application software design.

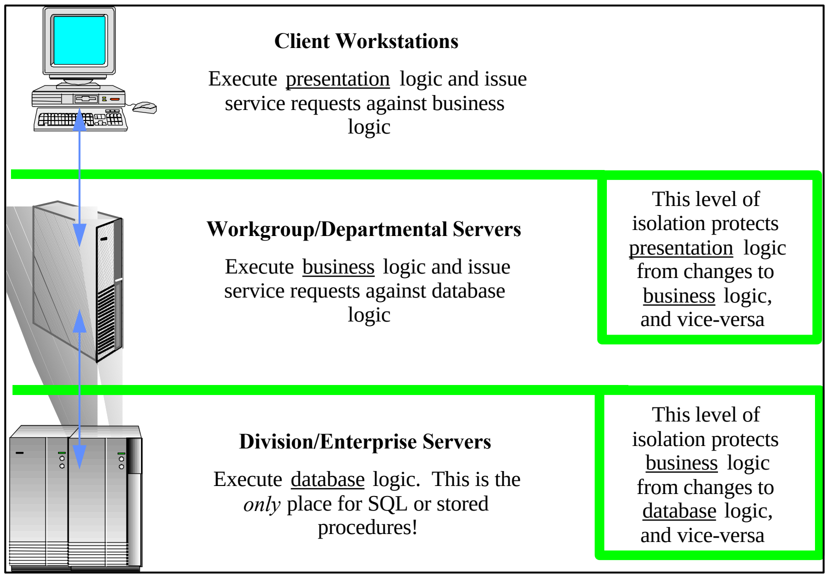
Client Workstations
Execute presentation logic and issue service requests against business logic
Workgroup/Departmental Servers
Execute business logic and issue service requests against database logic
This level of isolation protects presentation logic from changes to business logic, and vice-versa
Division/Enterprise Servers
Execute database logic. This is the only place for SQL or stored procedures!
This level of isolation protects business logic from changes to database logic, and vice-versa

Figure 2-9. Partition of Logical Software Distribution

In addition to the one-tier logical software distribution, which is synonymous with the monolithic physical software distribution, logical software distribution strategies that are commonly used to describe architectures are:

- **Two-tier logical distribution**, in which application logic is intertwined with either (or both) the presentation or the data management functions and delivered on the same platform. Not all software components are cohesive and they may not be loosely coupled. With the two-tier distribution model, the server either runs no application logic (i.e., runs only the DBMS) or embeds all server-based application logic within the DBMS. As a result, changes to the application logic often complicate maintenance of the presentation logic and/or the data management logic. The distributed data management and remote data management physical distribution models are typically implemented as two-tier distributions. In the recent past, distributed systems were almost exclusively two-tier. However, in many cases, these implementations have proven hard to modify, difficult to manage, expensive, and difficult to scale for enterprise-wide deployment. As a result, the three-tier strategy is often considered more appropriate for large, mission-critical systems.

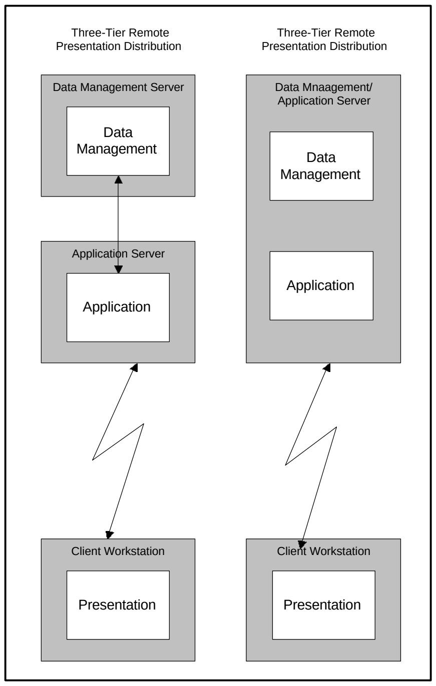

Figure 2-10. Logical Separation of Software

- **Three-tier (multi-tier) logical distribution** where presentation, application, and data management services are designed as separate components that can be delivered on different platforms. Because multi-tier applications are highly cohesive and loosely coupled, they can be delivered across multiple (three or more) physical tiers based on the available infrastructure, performance requirements, and standards. Most multi-tier applications distribute user-related I/O logic to the client platform, and most or all of the data-related I/O logic to server resources. However, application and data management logic may be distributed in any number of ways depending upon the availability of specialized servers and on other infrastructure considerations.

As is the case with physical software distribution strategies, an architecture may simultaneously employ several logical distribution strategies. Considerations for selecting a logical software distribution strategy for an architecture are presented in Figure 2-11.

	Strengths	Weaknesses
Two-tier	• Mature toolsets are available • Required skillsets and experience are common • Simpler approach to design • Generally simpler to implement. • More infrastructure independent • Uses the power of the client for a more robust user interface and application logic execution	• Business and presentation logic are not isolated or insulated from each other • Limited code reusability – there is no common application logic • Limited scalability and flexibility • Requires a more complex desktop or client • Software configuration distribution and management is more complex and harder to administer • More sensitive to network bandwidth • Generates higher network loads and stresses
Three-tier	• Middle tier provides expanded levels of service • Enhances scalability and flexibility • Easier to develop and support complex applications over time • Reduces network traffic between clients and application servers • Requires a less complex client • Easier to integrate multiple, heterogeneous data sources • Is the current industry direction for enterprise applications	• Requires a more complex environment • Built on emerging technology • Required skill-sets and experienced developers not common • Normally involves higher levels of integration and data distribution • Requires more up-front analysis and design

Figure 2-11. Strengths and Weaknesses for Logical Software Distribution Strategies

2.3.3 Data Distribution Strategies

It is common for an organization's various business units to require access to the same information. In the past, access to centralized data could not be guaranteed during network failures, resulting in less application availability, reduced data accessibility, downtime, lost revenue, and inconvenience. Distributed applications needing access to centralized data may suffer poor performance due to relatively limited wide area network bandwidth and throughput capabilities. In addition, competition among applications for the same data within a system can adversely affect performance and response time.

Data distribution and access strategies have a direct impact on application design, end user performance, and operational support. Long-term application growth and enhancement, as well as the need to support increased numbers of users, may dictate that data be distributed to a number of locations. Data distribution decisions also have an impact on the network topology, transport mechanisms, and bandwidth requirements of a system. As a result, data distribution strategies must be considered when designing a technical architecture.

The data distribution strategies commonly used to describe system architectures are:

- **Centralized data** in which all data is maintained in one central location, from which users access it. The centralized data strategy is the least complicated data distribution strategy and is the most secure. However, this strategy can adversely affect system scalability and introduces risk, as all distributed system processes are dependent upon the availability of a single resource – the centralized database (see Figure 2-12).

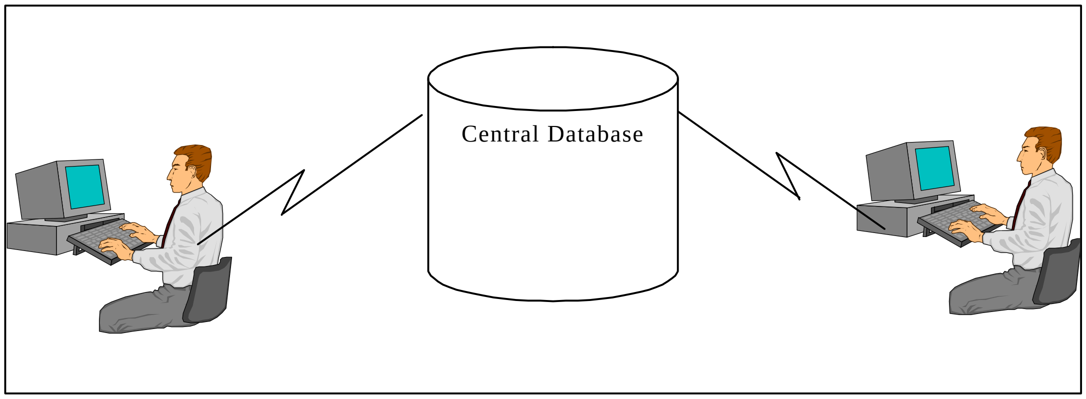

Figure 2-12. Centralized Data

- **Distributed data** in which data resources are distributed to multiple sites, where they can be used by similarly distributed system components. Data distribution configurations are complicated and can be difficult to maintain because business rules associated with data management must be maintained at more than one site.

  With data distribution, the same data schema and data is distributed to each target site. Data distribution provides targets sites with access to all of an organization's distributed data. However, this distribution strategy can substantially increase network loads and requires the use of two-phase commit mechanisms to ensure distributed data remains synchronized as changes are made at each target site. (Two-phase commit mechanisms ensure that updates are successfully made to all copies of the data before changes are committed to any of the copies.) Data distribution is illustrated in Figure 2-13.

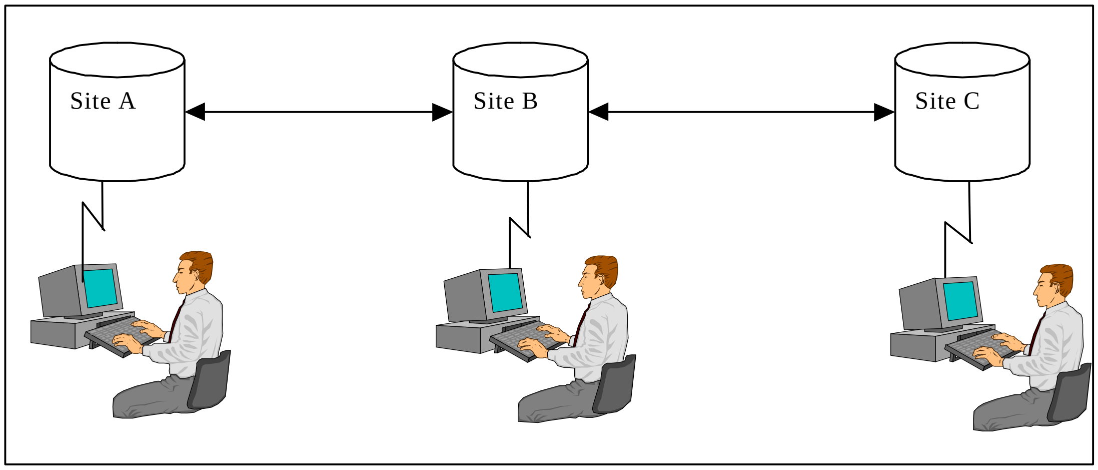

Figure 2-13. Distributed Data

- **Replicated data** in which data is copied to distributed target sites. Data replication can take two forms, (a) primary-site data replication for data publication and (b)primary-site data replication for data consolidation. Figure 2-14 illustrates primary-site replication for data publication and Figure 2-15 illustrates primary-site data replication for data consolidation. The central difference between these two configurations is that the "data publication" model only allows updates to the centralized data source, whereas the "data consolidation" model allows data at the distributed target sites to be updated.

With primary-site replication for data publication, the primary site copies data to multiple target data stores and data is changed only at the primary site. The most simplistic example of this model is a single primary site that replicates all its data to a secondary system or to a set of identical secondary systems. In a more complicated configuration, portions of the primary site database could be copied to specified secondary sites, with each secondary site potentially receiving a different portion of the primary site database. This data replication configuration is often used to create hot-site backups and to populate distributed decision support system data repositories.

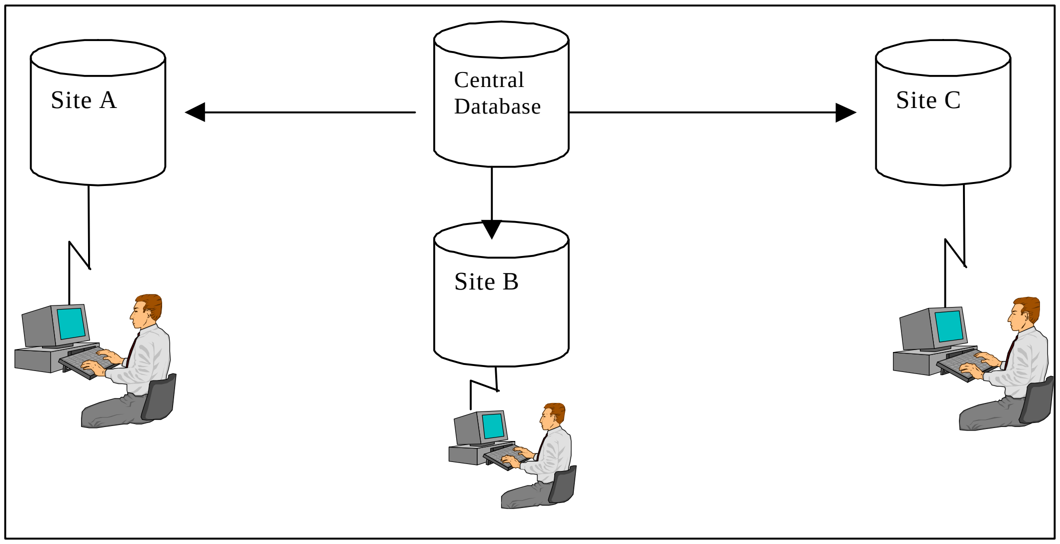

Figure 2-14. Distributed Data with Replication for Data Publication

With primary-site replication for data consolidation there may be multiple primary data resources, each of which replicated data to a centralized secondary site. Data consolidation may occur when the centralized secondary site "pulls" data from the remote primary sites. Alternatively, the remote primary sites may "push" data to the centrally located database. The "push" strategy is often preferred in situations involving mobile remote sites, which may not be consistently available to the centralized site. This "data consolidation" configuration is often useful in those situations where data may need to be regularly aggregated and reviewed, but distributed components need to be able to work without always being connected to the centralized site.

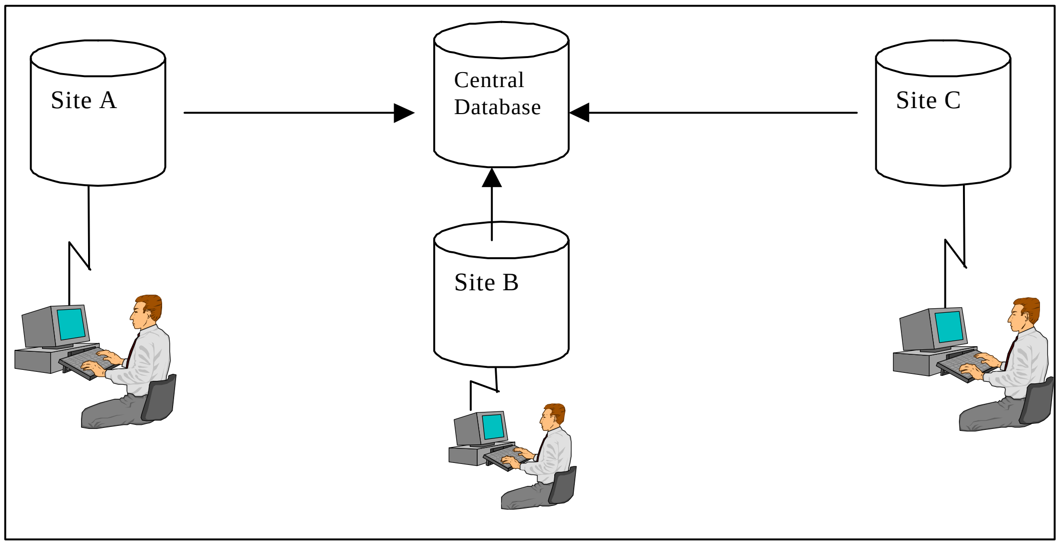

Figure 2-15. Distributed Data with Replication for Data Consolidation

As is the case with the physical and logical software distribution strategies, a technical architecture may simultaneously employ several data distribution strategies. When considering data distribution strategies the strengths and weaknesses outlined in Figure 2-16 should be considered.

	Strengths	Weaknesses
Centralized	• Easiest to maintain • Most secure • Eliminates data synchronization issues	• Hardest to scale • Can be performance constrained • Not flexible • Limited support for distributed autonomous operations
Distributed	• One view of the data for all users and applications • Can partition for performance • Tools provided by the DBMS vendor	• Require two-phase commit for data integrity and synchronization • Can be network intensive • Complex to setup and maintain
Replicated	• Allows for local variants • Very scaleable for performance • DBMS vendor provides tools for synchronization	• Can increase network loads • Data can be out of synchronization • Complicated to maintain

Figure 2-16. Strengths and Weaknesses of Data Distribution Strategies

2.4 Project EASI/ED Framework Architecture

As indicated in subsection 2.1, this document defines, evaluates, and recommends a framework technical architecture for Project EASI/ED. This framework technical architecture, which is described in Section 5, is primarily based on the physical software (process) and data distribution strategies described in subsection 2.3. Subsequent system architectures (conceptual and target) will use the logical software (process) distribution strategies to build upon the framework architecture.